Atty. Docket No. MS301930.1

CONCURRENT ARBITRATION OF INTERRUPT RESOURCES

by

Jacob Oshins

MAIL CERTIFICATION

I hereby certify that the attached patent application (along with any other paper referred to as being attached or enclosed) is being deposited with the United States Postal Service on this date **September 29, 2003**, in an envelope as "Express Mail Post Office to Addressee" Mailing Label Number **EV330022878US** addressed to the Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Gregory Turocy

Title: CONCURRENT ARBITRATION OF INTERRUPT RESOURCES

TECHNICAL FIELD

The present invention relates generally to computer systems, and more particularly to a system and method that employs multi-dimensional mapping of a range of potentially available interrupt resources that satisfy a given request.

BACKGROUND OF THE INVENTION

Modern operating systems drive many of today's technology-based innovations by offering a platform for both hardware and software development while serving many diverse needs. These systems have evolved from more simplistic file management systems to more complex workstations that provide high-end performance at reasonable cost. Such systems often include multi-processing architectures, high-speed memory, advanced peripheral devices, a variety of system libraries and components to aid software development, and intricate/interleaved bus architectures, for example. At the heart of these systems, include processor support functionality such as advanced memory controllers and interrupt controllers that route a plurality of system interrupts to one or more intelligent devices or components. As systems become more complex, such as in a multi-processing system, management of interrupt resources (*e.g.,* managing, processing, and allocating interrupt requests (IRQ)), has become ever more challenging.

In one example of a modern processing architecture, within a Plug and Play (PnP) subsystem employed in many computer systems, there is a concept of an "arbiter." This is generally a software plug-in, supplied by a Plug and Play Manager or a device driver, for example, which arbitrates resources for other devices. Resources are an aspect of Plug and Play that can be expressed as a range. Memory address space on a PCI bus, for example, can be expressed as a starting address and an ending address. In this example, a PCI driver provides an arbiter for memory address space on a PCI bus.

Interrupt resources (also known as IRQs) are arbitrated much like any other resource. Respective IRQs can be expressed as a range, usually as IRQ 1 through IRQ 10, for example. If a device needs an IRQ, it may request resources *via* PnP messages, referred to as IRPs. Thus, an associated arbiter decides which IRQ is appropriate, and assigns a specific resource for the task at hand. This typically operates in existing PnP subsystems, by assigning an IRQ first, and then determining other data surrounding the IRQ, such as an affinity mask, which is a set of processors that can be targeted by the interrupt. This attachment of extra information associated with the IRQ generally occurs in a "resource translation" phase of resource assignment.

Previous methods, which involved arbitrating an IRQ and then selecting associated data after the fact, was generally sufficient for earlier operating systems, but it was not always flexible enough for emerging I/O bus technologies. It also made it difficult for administrators to intelligently determine a set of processors that would be targeted by a specific device's interrupts. Also, the infrastructure in Plug and Play subsystems - as well as others, operates with device resources in a one-dimensional manner, wherein respective resources are treated separately.

Resources can be, but are not limited to, the following types: I/O Ports; Memory locations; IRQs; DMA channels; and Bus Numbers. While a device may need resources in multiple categories, respective resource requests are generally taken separately, and represented as a one-dimensional range. A device (A) may, for example, request a resource set that appears like the following:

Any 16 I/O ports;
Any 256 bytes of memory address space;
One IRQ between 0 and 255; and
Four bus numbers.

Or, a different device (B) may request:

I/O Port 0x220 through I/O port 0x221; and
IRQ 3 or IRQ 5.

The PnP subsystem generally breaks down the problem into separate resource requests and passes them onto the arbiter. The arbiter for I/O ports, would then receive a request such as:

Device A – any 16 I/O ports;

Device B – I/O Port 0x220 through I/O port 0x221.

The arbiter would then attempt to satisfy these requests. This one-dimensional view of resources is typically suitable for most resource types except IRQs. In order for a device to interrupt the processor, the device has to be connected to an input on an interrupt controller and it also has to insert an interrupt service routine (ISR) into an associated interrupt descriptor table (IDT). Thus, the choices involved in interrupt assignment are inherently two-dimensional.

This two-dimensionality was not completely understood when PnP subsystems in various operating systems were originally designed. In one case, an 8259 interrupt controller has a relatively fixed IRQ-to-IDT mapping. Generally, operating systems program a base IDT entry, 0x70 for example, for IRQ 0. Thus, IRQ 9 will utilize IDT entry 0x79. Furthermore, since the IDT exists separately for respective processors in the machine, if the machine has more than one processor, the IDT entry component of the interrupt assignment is itself multi-dimensional. Complicating the matter even more, new I/O bus technologies, summed up in the term Message-Signaled Interrupt, allow a device to trigger more than one IRQ, or sometimes cause a processor to jump through an IDT entry to an ISR without involving an IRQ.

These aspects, taken together, indicate that traditional approaches, which involve selecting an IRQ in the arbitration process and then selecting IDT entries at a later time, may cause operating system problems. One factor is that there may not be enough IDT entries available, or the ones that are available may be associated with processors that are not optimal. Thus, the arbiter may answer the PnP manager indicating that there is an IRQ that is applicable for a device, but after assignment is achieved, the device still would not be able to deliver an interrupt.

SUMMARY OF THE INVENTION

The following presents a simplified summary of the invention in order to provide a basic understanding of some aspects of the invention. This summary is not an extensive overview of the invention. It is not intended to identify key/critical elements of the invention or to delineate the scope of the invention. Its sole purpose is to present some concepts of the invention in a simplified form as a prelude to the more detailed description that is presented later.

The present invention relates to systems and methods that provide multi-dimensional mapping, dynamic management, and control of interrupt resources in a concurrent manner. In one aspect, one or more interrupt resources are requested by a requesting component, wherein the interrupt resources relate to system interrupt assignments and their associated interrupt descriptor table mappings. The interrupt assignments generally relate to inputs, sources, or events that cause processors to exit from current code execution in order to service code relating to the interrupt, whereas the descriptor table provides memory locations that indicate where (in memory) service or code execution is to commence when servicing the interrupt.

An arbiter receives the request for resources, and essentially answers yes or no as to whether or not the request can be satisfied. For example, the requesting component may request three interrupt resources, wherein the arbiter satisfies the request, if possible, by returning interrupt assignments 0 through 2 (or other interrupt range), and also returns associated descriptor table mappings for the respective interrupt assignments (*e.g.,* 0x090 through 0x092), thus facilitating that both an interrupt and descriptor table can be mapped and ultimately executed. In this manner, interrupts and respective mappings are dynamically assigned/determined during a negotiation phase between the requesting component and the arbiter resulting in a multi-dimensional mapping of interrupts and their respective descriptor table assignments. In contrast to conventional one-dimensional systems, that merely assign an interrupt and have resultant descriptor tables map by default to the assignment or assigned at a later time, the present invention provides a flexible two (or more) dimensional mapping that facilitates execution of

interrupts in more complex systems such as multi-processor/multi-bus operating systems while mitigating possible uncertainty or execution problems involved with assigning an interrupt and selecting a descriptor table entry by default or at a later time.

In one particular aspect of the present invention, a contiguous device interrupt arbitration system and process is provided that integrates arbitration of one or more Interrupt Requests (IRQ) for a device with other I/O bus resource interrupts. Such interrupts can be generated from substantially any device or component and include such aspects as message signaled interrupts that can allow a given device to generate more than one IRQ. Various system buses can also be employed and managed that include local processor buses, system-wide buses as well as more specialized buses such as PCI and ISA buses. As noted above, the present invention replaces prior serial or one-dimensional interrupt handling of relatively fixed IRQ-to-IDT resource mapping with a more flexible two-dimensional mapping of a range of resources that are potentially available to satisfy a given request, wherein an IDT is an Interrupt Descriptor Table that identifies sections of code that run in response to a particular IRQ. Thus, the present invention essentially merges arbitration for an IRQ with an arbitration of other interrupt-related data in order that other features become possible such as:

- Intelligent assignment and management of target processor sets for interrupts;
- Assignment of interrupts that employ IDT entries, but not interrupt controller inputs;
- Assignment of interrupts that will be delivered on processors that are physically close to devices that generate the respective interrupts;
- The ability to control or dynamically tune a system based upon monitored and/or changing conditions and influencing interrupt assignment policy; and/or
- Optimization for non-uniform memory architectures.

To the accomplishment of the foregoing and related ends, certain illustrative aspects of the invention are described herein in connection with the following description and the annexed drawings. These aspects are indicative of various ways in which the

invention may be practiced, all of which are intended to be covered by the present invention. Other advantages and novel features of the invention may become apparent from the following detailed description of the invention when considered in conjunction with the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic block diagram of an interrupt arbitration system in accordance with an aspect of the present invention.

Fig. 2 is a diagram of a multi-component arbitration system in accordance with an aspect of the present invention.

Fig. 3 is a schematic block diagram illustrating an example multi-bus system in accordance with an aspect of the present invention.

Fig. 4 is a schematic block diagram illustrating an example processor and interrupt controller in accordance with an aspect of the present invention.

Figs. 5 and 6 illustrate an exemplary interrupt arbitration system in accordance with an aspect of the present invention.

Fig. 7 is a schematic block diagram illustrating an example operating system environment in accordance with an aspect of the present invention.

Figs. 8-11 are flow diagrams illustrating an example arbitration process in accordance with an aspect of the present invention.

Fig. 12 is a schematic block diagram illustrating a suitable operating environment in accordance with an aspect of the present invention.

Fig. 13 is a schematic block diagram of a sample-computing environment with which the present invention can interact.

DETAILED DESCRIPTION OF THE INVENTION

The present invention relates to a system and methodology to facilitate negotiation, assignment, and management of interrupt resources in a flexible and dynamic manner. An interrupt arbitration system is provided to process at least one request associated with an interrupt resource, wherein the request includes at least two dimensions related to an interrupt and an interrupt service component such as an interrupt descriptor table entry, for example. An arbiter processes the request and returns a subset of interrupt resource ranges in view of available system resources. This multi-dimensional mapping of resources enables coordination across resource pools, processors, buses, and/or other components while mitigating possible run-time problems attributed to one-dimensional systems that may find that suitable resources are unavailable. By enabling an arbitration process to be multi-dimensional, respective system arbiters can be essentially certain that resource assignments are viable before committing to an interrupt assignment.

As used in this application, the terms "component," "requestor," "arbiter," "system," and the like are intended to refer to a computer-related entity, either hardware, a combination of hardware and software, software, or software in execution. For example, a component may be, but is not limited to being, a process running on a processor, a processor, an object, an executable, a thread of execution, a program, and/or a computer. By way of illustration, both an application running on a server and the server can be a component. One or more components may reside within a process and/or thread of execution and a component may be localized on one computer and/or distributed between two or more computers.

The following terms are also defined for purposes of clarity such as defining one or more acronyms that may be employed as follows:

Interrupt – An action taken by an I/O device that causes the processor to start servicing the device. The code that the processor executes is called an "Interrupt Service Routine," or ISR, and it is part of an associated device driver.

Interrupt Controller – A device within a computer that presents interrupts to a processor. It is attached concurrently to respective I/O devices and to the processor.

IRQ – Interrupt Request. This term can be employed in various manners. It generally denotes a number associated with a distinct interrupt controller input.

IDT – Interrupt Descriptor Table. In an Intel x86 processor, for example, there is a table in memory that lists 256 different portions of code that can be run in response to an interrupt. In response to an interrupt, the processor receives a number between 0 and 255 from the interrupt controller and then jumps to the code listed by the respective entry in the table. In non x86-processors, this behavior can be (and usually is) emulated in software. Each processor in a single machine generally has a distinct IDT.

IDT Entry – Single listing in the IDT, usually associated with a specific IRQ.

8259 PIC – Traditional Programmable Interrupt Controller that typically exists in all personal computers (PCs). It has 16 IRQs. It generally cannot operate in multi-processor machines.

APIC – Advanced Programmable Interrupt Controller. A full APIC implementation involves parts distributed throughout a machine. If these parts exists, multi-processor operation becomes possible. APIC systems are not limited to 16 IRQs.

Local APIC – Part of an APIC system that exists within a processor.

I/O APIC – Part of an APIC system that collects interrupts from devices. Exists outside the processor.

Message-Signaled Interrupt (MSI) – This is an interrupt that is delivered as a write cycle (or perhaps a packet) on an I/O bus, and not through a dedicated wire.

Referring initially to Fig. 1, an interrupt arbitration system 100 is illustrated in accordance with an aspect of the present invention. A requestor 110 (or requesting component) communicates over a bus 120 to an arbiter 130 in order to determine available system resources that may be utilized by the requestor to perform one or more tasks. Such resources may be related to I/O ports, memory locations, DMA channels (Direct Memory Access), bus numbers, and/or interrupt requests (IRQs), wherein arbiters typically exist for each type of resource. These resources may be associated with one or

more system devices, buses, and/or other components illustrated at 140 and which are negotiated for *via* the arbiter 130.

In one aspect, the requestor 110 issues a multidimensional interrupt request (IRQ) 150 associated with one or more interrupt inputs, signals, or assignments, whereby the IRQ is employed to interrupt a processing component (not shown) which is described in more detail below. Along with the request 150, a respective Interrupt Descriptor Table (IDT) entry or data can also be requested in order that the IRQ and IDT can be concurrently mapped, wherein the IDT provides a memory location where interrupt service routines can be executed when a processor receives an associated IRQ. Although the IRQ and IDT data are generally requested in a concurrent manner (*e.g.*, within the same request), it is to be appreciated that these components (IRQ and/or IDT) can be requested on separate requests or queries to the arbiter 130.

Upon receiving the request 150, the arbiter 130 performs an analysis of system resources. If the requested interrupt resources (ISR and IDT) are found to be available, the arbiter 130 returns a resource subset or data packet 160 indicating the resources that can be utilized by the requestor 110. If the resources are not deemed available by the arbiter 130, a code or flag may be returned indicating the requested resources cannot be satisfied. As can be appreciated further requests and arbitrations can subsequently be negotiated. Moreover, other dimensions can be provided for and negotiated in the request 150. For example, a time element could be provided in the request 150, wherein the requested resources are negotiated for specified periods (*e.g.*, Request X interrupts, having Y IDT ranges, for period of less than 1 second – or other time). After the designated period, the resources can be relinquished by the requesting component and negotiated for by other components.

In one specific example of the system 100, the requestor 110 can be a Plug and Play (PnP) manager that communicates with individual plug-in modules referred to as arbiters 130, which decide which resources can be assigned to specific devices. These arbiters are generally not within the PnP manager itself since the PnP manager doesn't typically have internal knowledge of specific bus technologies. The PnP manager does

not, for example, know the specifics of selecting I/O ports for devices on a PCI bus. A PCI driver, however, typically keeps track of such details. Thus, the PCI driver generally supplies the arbiter 130. It is to be appreciated that the arbiter 130 can be provided as a global resource and thus arbitrate over various system buses, components, devices, and/or other resources.

Continuing the above PnP example, arbiters typically communicate with the PnP manager using a standard interface. The interface definition generally includes functions for the following:

- Testing/analyzing whether a possible set of resources can work.
- Committing a specific set of resources that has been requested by the PnP manager.
- Querying for the set of devices that might conflict with a resource set.
- Marking resources that were already in use by a BIOS (or other component) when a machine booted.

Much of the arbiter's work is provided in a "Test" interface. It essentially answers a yes or no question posed by the PnP manager or requestor 110. The PnP manager queries "can this possible set of resources coexist, and if there are any ambiguities in the resources set, how should they be resolved?" The arbiter 130 answers "no," or it resolves the ambiguities and answers "yes" (*e.g.*, "Yes, Device B can have I/O Ports 0x220 and 0x221 and Device A should use I/O ports 0x2000 through 0x200f.") Typically, arbiters can be implemented using a core library of functions that are centered around resource manipulations. However, the present invention can also utilize this library, for simplicity of implementation and also process many operations, including the multi-dimensionality of IRQ resources, outside of the base library. It is also possible to create an arbiter that functioned with the PnP manager, but which didn't utilize the library.

A function can be created within the library, called FindSuitableRange, that the library calls in order to find a range of available resources. In one implementation (in a

memory arbiter, for example,) FindSuitableRange searches for an unclaimed range in the list of available resources. In the interrupt resource aspect, FindSuitableRange searches across available IRQs and available IDT entries.

Various features can be supported with the above infrastructure and include:

1. A component for an administrator (or other control software) to tune the performance of the machine (*e.g.*, PC) by influencing interrupt assignment policy. For example, a control component could monitor system performance over time. Different interrupt resource assignments could then be monitored in view of system performance in order that resources can be re-allocated (automatically and/or manually) and executed in a substantially optimum manner.
2. Message-Signaled Interrupts (MSI).
3. Optimization for Non-Uniform Memory Architecture machines.

Application Programming Interfaces (APIs) can also be provided to expose these features. The following exist for interrupt management:

- A component for influencing IRQ target processor policy and IRQ priority policy.
- A component for conveying policy and other IRQ-related data to and from an IRQ arbiter.

The following exist for manipulating interrupt hardware:

- A component for collecting data to enable MSI in hardware. - IRQ_ARBITER_INTERFACE
- A method to request that an Interrupt Message Service Routine be inserted into the processor's IDT. – IoConnectInterruptEx
- A method to request that an Interrupt Message Service Routine be removed from a processor's IDT. – IoDisconnectInterruptEx

Referring now to Fig. 2, a multi-component arbitration system 200 is illustrated in accordance with an aspect of the present invention. The system 200 illustrates a logical configuration relating to how arbitration can occur between a plurality of requesting

components and arbiter components distributed across a multi-bus and/or multi-processor system. In general, requests pass from one software component (or hardware component) to another, and are not generally associated with any bus. In contrast, devices generally exist on a bus, wherein a bus driver may employ bus mechanisms to discover enough about the device to produce requests that will be passed to the arbiter. Thus, devices have properties which, when recognized by the device driver or bus driver, can be converted by driver software into a request that will ultimately be presented to an arbiter, which is typically another software component.

As noted above, the system 200 depicts a logical configuration of components participating in an arbitration process. In this aspect, one or more buses (1-B) at 210 can be employed to provide communications between one or more requestors at 220 (1-R) and one or more arbiters 230 (1-A), wherein A, B, and R are integers respectively. For example, one bus 210 may represent a local bus, system bus, and/or a more specialized bus such as a PCI bus or ISA bus. Requestors 220 may communicate with one or more other requestors and/or arbiters 230 across the buses 210. Likewise, arbiters 230 may communicate with one or more other arbiters and/or requestors across the buses 210. It is to be appreciated that interrupts can be derived from a plurality of sources and components.

After arbitration has occurred, and system interrupts are functional in accordance with the multi-dimensional mapping described above, various components associated with the respective buses may issue interrupts to respective processors. In one example, devices associated with a first bus (*e.g.,* ISA bus) could have interrupts that are related to devices on a second bus (*e.g.,* PCI bus). Such a configuration can exist, wherein interrupts from respective buses are wired-OR together and provided to a processor as a single interrupt. Thus, if a processor were to be interrupted from interrupt 4, for example, and interrupt 4 was an output from an OR gate having interrupt signals from separate buses, then the processor would likely perform polling procedures to determine which bus the interrupt was actually generated from. After servicing the interrupt, the processor would likely perform other polling procedures to determine that respective devices on the

associated bus had not attempted to generate another interrupt or that all interrupts had been properly serviced (*e.g.*, if both buses interrupted on interrupt 4, another service routine would then likely be executed).

Turning to Fig. 3, a system 300 illustrates an example multi-bus system in accordance with an aspect of the present invention. In this aspect, a central processing unit (CPU) 310 communicates *via* a north bridge adaptor 320 to a system bus 330. The north bridge adaptor also provides access to other resources such as memory 340. As illustrated, two PCI bridges 350 and 360 are provided that communicate with the bus 330. These bridges link to PCI buses 1 and 2 respectively. Also, an ISA bridge 370 operatively couples the bus 330 and ISA bus 1. As can be appreciated a plurality of other buses, adaptors, bridges, and/or other components can be provided that operate with the CPU 310 (or CPUs). Such components can generate interrupts for the CPU in accordance with the multidimensional arbitration systems described herein.

Fig. 4 is a schematic block diagram illustrating an example processor and interrupt controller in accordance with an aspect of the present invention. A portion of a processor 410 is depicted having a local Advanced Programmable Interrupt Controller (APIC) 420 that receives/processes interrupts directed to the processor 410. This can include an interrupt request register, an in-service register indicating which interrupt is currently being serviced, and a trigger mode register. The APIC 420 can receive interrupts directly connected to the processor 410 and/or from indirect sources such as from an I/O APIC 430 that collects interrupts from devices outside the processor 410 or local processing system. As illustrated, the I/O APIC 430 can include inputs for servicing a plurality of interrupts and include functions such as edge or level triggering options for the inputs, pin assertion functions, as well as other programming options. It is to be appreciated that the multidimensional arbitration described above can be employed with substantially any type of processing unit and/or interrupt controller in addition to the example components described herein.

Figs. 5 and 6 illustrate an exemplary interrupt arbitration system in accordance with an aspect of the present invention. A multiprocessor architecture is depicted having

a processor at 510 and 520, wherein the respective processors are shown with associated IDT range entries and possible IDT range entries which are not presented to an arbiter system library (not shown). In addition the processor may interact with interrupt controllers at 530 that receive interrupt inputs from optional link nodes at 540, wherein the link nodes operate as interrupt multiplexers that connect multiple interrupt sources to a single interrupt on the interrupt controllers 530. The link nodes 540 receive input from one or more IRQ user data sources at 550 of Fig. 5 and at 560 From Fig. 6. These sources contain such information as processor information, interrupt controller information, link node connection information, and MSI (Message-Signaled Interrupt) target information. As illustrated the sources at 550 are associated with allocation ranges depicted at 570 of Fig. 6, and the sources at 570 of Fig. 6, are associated with possible allocation ranges at 580 of Fig. 6.

Fig. 7 illustrates an example operating system environment 700 in accordance with an aspect of the present invention. In the system 700, an application 710 having a configuration manager 714 for device-specific data, and a Windows Management Infrastructure (WMI) component 718 for machine-specific data interact via remote procedure calls (RPC) with a COM server at 730 and a PnP manger service at 734. These components 730 and 734 interact across a user mode/kernel mode boundary 740 with a kernel-mode WMI manager 750 and a kernel-mode PnP manager 754. The manager's 750 and 754 interact with an IRQ arbiter 760 that is associated with a driver component 764. As described above, interrupt resources are negotiated for in a multidimensional manner. This includes negotiating in conjunction with the IRQ arbiter 760 and respective components that may act directly and/or indirectly with the arbiter.

Figs. 8-11 are flow diagrams illustrating an example arbitration process in accordance with an aspect of the present invention. While, for purposes of simplicity of explanation, the methodologies are shown and described as a series of acts, it is to be understood and appreciated that the present invention is not limited by the order of acts, as some acts may, in accordance with the present invention, occur in different orders and/or concurrently with other acts from that shown and described herein. For example,

those skilled in the art will understand and appreciate that a methodology could alternatively be represented as a series of interrelated states or events, such as in a state diagram. Moreover, not all illustrated acts may be required to implement a methodology in accordance with the present invention.

Before proceeding, it is noted that Fig.8 generally applies to a determination between ISA device processing, Message-Signaled Interrupt processing, and PCI device processing, wherein further ISA device processing is illustrated in Fig. 9 and further PCI device processing is illustrated in Figs. 10 and 11, respectively.

Proceeding to 810 of Fig. 8, a set of resources is requested by a device. At 820, a decision is made as to how the requesting device delivers interrupts. If an ISA device is detected at 820, the process proceeds to 830, wherein a determination is made regarding the devices interrupt such as edge triggered of active low, for example. Also, a determination is made as to whether there is a free IDT entry. Additional ISA device processing is described with respect to Fig. 9 below.

If a PCI bus device is detected at 820, the process proceeds to 840, wherein a determination is made whether the device a connected to an interrupt controller or an IRQ link node described above. If the device is connected to an interrupt controller at 840, the process proceeds to Fig. 11. If the device is connected to a link node, the process proceeds to Fig. 10.

If message-signaled interrupts are determined at 820, the process proceeds to 850, wherein various procedures occur. Such procedures include determining which processors the device's interrupt should be connected to, determining how many IDT entries will be needed, and searching a set of target processors' IDTs for a group of IDT entries that are available on the processors in the set. At 860, a determination is made as to whether such a group of entries exist. If not, the process proceeds to 870 and ends without satisfying the request. If yes, the process proceeds to 880 and assigns a "dummy" or stub IRQ and a set of IDT entries.

With reference to Fig. 9, further ISA device processing is illustrated. At 910, an IRQ is selected form the ISA device's possible resources. At 920 a determination is

made as to whether any IRQs are left in the possible resources. If not, the process ends at 930 without satisfying the request. If so the process proceeds to 940. At 940, a determination is made as to whether an IRQ is already in use. If not, the process assigns this IRQ and a free IDT entry at 950. If yes, the process proceeds to 960. At 960, the process determines whether properties of the device that is already using the IRQ matches the properties of the new device. If not, the process proceeds back to 910 and selects another IRQ. If so the process proceeds to 970. At 970, the process determines whether both the existing device and the new device support sharing IRQs. If not, the process proceeds back to 910 and selects another IRQ. If so, the IRQ is assigned along with a free IDT entry at 980.

Proceeding now to Fig. 10, further PCI device processing for link nodes is illustrated. At 1010, a determination is made as to whether a link node is in use. If so the process proceeds to 1020 and determines whether the IRQ that is currently in use for this link node is within the set of possible resources. If not, the process proceeds back to 870 of fig. 8 and ends. If yes, the process assigns the IRQ and IDT entry at 1030 associated with the respective link node at 1030.

If the link node was not in use at 1020, the process proceeds to 1040 and determines which processors to connect the device interrupts to. At 1050, a determination is made as to whether IDT entry is available that is common to all processors. If not, the process ends at 1060. If yes, the process proceeds to 1070. At 1070 a determination is made as to whether there is an intersection between the possible resources and IRQs that the link node can use. If so, the IRQ and IDT entry to the link node and the device is assigned at 1080. If not, the process ends at 1060.

Proceeding to Fig. 11, further PCI device processing is provided for devices connected directly to an interrupt controller. At a determination is made as to whether a static IRQ is already in use. If not the process proceeds to 1120 and assigns an IRQ and IDT entry associated with a static vector. If yes at 1110, the process proceeds to 1130 and determines which processors the device's interrupts should be connected to. At 1140, a determination is made as to whether a free IDT entry is common to the respective

processors. If not, the process ends at 1150 without assigning resources. If yes at 1140, the process assigns the IDT set of entries to the IRQ and the device at 1160.

With reference to Fig.12, an exemplary environment 1210 for implementing various aspects of the invention includes a computer 1212. The computer 1212 includes a processing unit 1214, a system memory 1216, and a system bus 1218. The system bus 1218 couples system components including, but not limited to, the system memory 1216 to the processing unit 1214. The processing unit 1214 can be any of various available processors. Dual microprocessors and other multiprocessor architectures also can be employed as the processing unit 1214.

The system bus 1218 can be any of several types of bus structure(s) including the memory bus or memory controller, a peripheral bus or external bus, and/or a local bus using any variety of available bus architectures including, but not limited to, 16-bit bus, Industrial Standard Architecture (ISA), Micro-Channel Architecture (MSA), Extended ISA (EISA), Intelligent Drive Electronics (IDE), VESA Local Bus (VLB), Peripheral Component Interconnect (PCI), Universal Serial Bus (USB), Advanced Graphics Port (AGP), Personal Computer Memory Card International Association bus (PCMCIA), and Small Computer Systems Interface (SCSI).

The system memory 1216 includes volatile memory 1220 and nonvolatile memory 1222. The basic input/output system (BIOS), containing the basic routines to transfer information between elements within the computer 1212, such as during start-up, is stored in nonvolatile memory 1222. By way of illustration, and not limitation, nonvolatile memory 1222 can include read only memory (ROM), programmable ROM (PROM), electrically programmable ROM (EPROM), electrically erasable ROM (EEPROM), or flash memory. Volatile memory 1220 includes random access memory (RAM), which acts as external cache memory. By way of illustration and not limitation, RAM is available in many forms such as synchronous RAM (SRAM), dynamic RAM (DRAM), synchronous DRAM (SDRAM), double data rate SDRAM (DDR SDRAM), enhanced SDRAM (ESDRAM), Synchlink DRAM (SLDRAM), and direct Rambus RAM (DRRAM).

Computer 1212 also includes removable/non-removable, volatile/non-volatile computer storage media. Fig. 12 illustrates, for example a disk storage 1224. Disk storage 1224 includes, but is not limited to, devices like a magnetic disk drive, floppy disk drive, tape drive, Jaz drive, Zip drive, LS-100 drive, flash memory card, or memory stick. In addition, disk storage 1224 can include storage media separately or in combination with other storage media including, but not limited to, an optical disk drive such as a compact disk ROM device (CD-ROM), CD recordable drive (CD-R Drive), CD rewritable drive (CD-RW Drive) or a digital versatile disk ROM drive (DVD-ROM). To facilitate connection of the disk storage devices 1224 to the system bus 1218, a removable or non-removable interface is typically used such as interface 1226.

It is to be appreciated that Fig 12 describes software that acts as an intermediary between users and the basic computer resources described in suitable operating environment 1210. Such software includes an operating system 1228. Operating system 1228, which can be stored on disk storage 1224, acts to control and allocate resources of the computer system 1212. System applications 1230 take advantage of the management of resources by operating system 1228 through program modules 1232 and program data 1234 stored either in system memory 1216 or on disk storage 1224. It is to be appreciated that the present invention can be implemented with various operating systems or combinations of operating systems.

A user enters commands or information into the computer 1212 through input device(s) 1236. Input devices 1236 include, but are not limited to, a pointing device such as a mouse, trackball, stylus, touch pad, keyboard, microphone, joystick, game pad, satellite dish, scanner, TV tuner card, digital camera, digital video camera, web camera, and the like. These and other input devices connect to the processing unit 1214 through the system bus 1218 *via* interface port(s) 1238. Interface port(s) 1238 include, for example, a serial port, a parallel port, a game port, and a universal serial bus (USB). Output device(s) 1240 use some of the same type of ports as input device(s) 1236. Thus, for example, a USB port may be used to provide input to computer 1212, and to output information from computer 1212 to an output device 1240. Output adapter 1242 is

provided to illustrate that there are some output devices 1240 like monitors, speakers, and printers, among other output devices 1240, that require special adapters. The output adapters 1242 include, by way of illustration and not limitation, video and sound cards that provide a means of connection between the output device 1240 and the system bus 1218. It should be noted that other devices and/or systems of devices provide both input and output capabilities such as remote computer(s) 1244.

Computer 1212 can operate in a networked environment using logical connections to one or more remote computers, such as remote computer(s) 1244. The remote computer(s) 1244 can be a personal computer, a server, a router, a network PC, a workstation, a microprocessor based appliance, a peer device or other common network node and the like, and typically includes many or all of the elements described relative to computer 1212. For purposes of brevity, only a memory storage device 1246 is illustrated with remote computer(s) 1244. Remote computer(s) 1244 is logically connected to computer 1212 through a network interface 1248 and then physically connected *via* communication connection 1250. Network interface 1248 encompasses communication networks such as local-area networks (LAN) and wide-area networks (WAN). LAN technologies include Fiber Distributed Data Interface (FDDI), Copper Distributed Data Interface (CDDI), Ethernet/IEEE 1102.3, Token Ring/IEEE 1102.5 and the like. WAN technologies include, but are not limited to, point-to-point links, circuit switching networks like Integrated Services Digital Networks (ISDN) and variations thereon, packet switching networks, and Digital Subscriber Lines (DSL).

Communication connection(s) 1250 refers to the hardware/software employed to connect the network interface 1248 to the bus 1218. While communication connection 1250 is shown for illustrative clarity inside computer 1212, it can also be external to computer 1212. The hardware/software necessary for connection to the network interface 1248 includes, for exemplary purposes only, internal and external technologies such as, modems including regular telephone grade modems, cable modems and DSL modems, ISDN adapters, and Ethernet cards.

Fig. 13 is a schematic block diagram of a sample-computing environment 1300 with which the present invention can interact. The system 1300 includes one or more client(s) 1310. The client(s) 1310 can be hardware and/or software (*e.g.*, threads, processes, computing devices). The system 1300 also includes one or more server(s) 1330. The server(s) 1330 can also be hardware and/or software (*e.g.*, threads, processes, computing devices). The servers 1330 can house threads to perform transformations by employing the present invention, for example. One possible communication between a client 1310 and a server 1330 may be in the form of a data packet adapted to be transmitted between two or more computer processes. The system 1300 includes a communication framework 1350 that can be employed to facilitate communications between the client(s) 1310 and the server(s) 1330. The client(s) 1310 are operably connected to one or more client data store(s) 1360 that can be employed to store information local to the client(s) 1310. Similarly, the server(s) 1330 are operably connected to one or more server data store(s) 1340 that can be employed to store information local to the servers 1330.

What has been described above includes examples of the present invention. It is, of course, not possible to describe every conceivable combination of components or methodologies for purposes of describing the present invention, but one of ordinary skill in the art may recognize that many further combinations and permutations of the present invention are possible. Accordingly, the present invention is intended to embrace all such alterations, modifications and variations that fall within the spirit and scope of the appended claims. Furthermore, to the extent that the term "includes" is used in either the detailed description or the claims, such term is intended to be inclusive in a manner similar to the term "comprising" as "comprising" is interpreted when employed as a transitional word in a claim.